EXHIBIT 6

WHITHAM, CURTIS, CHRISTOFFERSON & COOK, P.C.
INTELLECTUAL PROPERTY LAW

Providing Global Intellectual Property Strategies & Solutions

March 23, 2012

BY EMAIL AND FEDEX

Paul J. Tanck, Esq.
Chadbourne & Parke, LLP
30 Rockefeller Plaza
New York, N.Y. 10112

Re: *Rockwell Automation, Inc. v. WAGO Corporation*,
Case No. 3:10CV718-WMC (W.D. Wis.)

Dear Paul:

Enclosed please find Defendants' Disclosure of Liability Expert as to Patent Non-Infringement and the Supplemental Declaration of Richard Hooper, Ph.D., P.E., which are being served pursuant to the Court's Preliminary Pretrial Conference Order in the above-referenced matter. (Dkt. 22, ¶ 4) We are providing a CD containing the references identified in Appendix A to Dr. Hooper's Declaration by FedEx for delivery on Monday March 26, and we will also attempt to serve those references by email in the file "References.zip." One of the references is attached to the report, as well.

Very truly yours,

Robert N. Cook

Robert N. Cook

Enclosures

IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF WISCONSIN

ROCKWELL AUTOMATION, INC. and ROCKWELL AUTOMATION TECHNOLOGIES, INC.,	'	Case No. 3:10CV718-WMC
Plaintiffs,	'	
v.	'	
WAGO CORPORATION and WAGO KONTAKTTECHNIK GMBH & CO. KG,	'	
Defendants.	'	

SUPPLEMENTAL DECLARATION OF RICHARD HOOPER, PH.D., P.E.

I, Richard Hooper, Ph.D., P.E., make the following declaration in lieu of affidavit pursuant to Section 1746 of Title 28 of the United States Code, 28 U.S.C. § 1746:

Introduction

1. This declaration supplements my declaration dated February 23, 2012, by addressing infringement issues and by supplementing my February 23 discussion of invalidity issues. I incorporate my February 23 declaration (sometimes referred to as a "report" or "expert report") by reference as if fully restated in the text of this declaration. My qualifications in this matter are detailed in my previous report on this Case and are incorporated by reference. This report does not address Count One (U.S. Patent No. 6,745,090) or Count Four (U.S. Patent No. 7,058,461) based on my understanding that they have been dropped from this Case.

2. I have physically inspected numerous WAGO 750, 758 and 767 series products and reviewed numerous manuals and WAGO 759 series application software. I have also reviewed and studied the report, and supplements, submitted by Arthur Zatarain (Zatarain) in this Case. I spoke by telephone with Mark DeCramer, the manager of the product support department of WAGO Corporation. See Appendix A to this report for a complete listing of materials reviewed. The materials listed in Appendix A are incorporated into this declaration by reference.

3. It is my understanding that a patent is infringed if even one claim of the patent is infringed and that a patent claim is infringed if, on comparing the properly construed claim to the accused

product or method, the patent owner succeeds in showing that every limitation of the patent claim is found in the accused product or method. It is also my understanding that a dependent patent claim, which incorporates the substance of another claim by reference, cannot be infringed unless the claim whose substance is incorporated by reference is also infringed.

Count Two: '232 Patent

4. I understand Plaintiffs are asserting Claims 1, 2, 3, 5, 10, 11, and 14 of U.S. Patent No. 6,745,232 (the '232 patent).

5. Zatarain (page 42) opines: "I do not believe any of the claim terms in the '232 patent require explanation or a Court's interpretation, as the terms are clear and easily understood not only by a PHOSITA, but simply by laypeople as well." This opinion is not correct. A number of terms used in this patent would not be clear and easily understood by a layperson. "Communications medium," "mode change message," and "step mode" are examples of terms which would not be easily understood by a layperson and which are relevant to my opinion in this analysis (but not the only examples of terms in the '232 patent requiring interpretation before a layperson could understand them).

6. Zatarain opines Wago Programmable Logic Controllers (PLCs) infringe the asserted claims of the '232 patent and bases this conclusion primarily on the operation of WAGO software product 759-333 of 2007 (CoDeSys 2.3 (2007)).

7. As detailed below, the same functionality cited by Zatarain in the CoDeSys 2.3 (2007) software was present in the prior art software products of CoDeSys 1.5 of 1997 (CoDeSys 1.5 (1997)) and WAGO 759-120 of 1999 (WAGO-I/O-PRO (1999)).

8. It is my understanding that Codesys 1.5 (1997) and WAGO-I/O-PRO (1999) precede the August 23, 2000 filing date of the '232 patent.

Analysis Claim 1

9. The table below shows both CoDeSys 1.5 (1997) and WAGO-I/O-PRO (1999) include the same functionality cited by Zatarain from CoDeSys 2.3 (2007) in his analysis of Claim 1 of the '232 patent.

Cross reference of Claim 1 elements to CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007)			
	CoDeSys 1.5 (1997)	**WAGO-I/O-PRO (1999)**	**CoDeSys 2.3 (2007)**
A method for performing a function in a control device…	X	X	X
storing a program in a control device…	X	X	X
executing at least a portion of the stored program…	X	X	X
suspending execution of the stored program…	X	X	X
receiving a mode change message…	X	X	X

10. **Claim 1 - A method of performing a function in a control device comprising:**

11. While some WAGO-I/O systems may include a method for performing a function in a control device, as discussed below, the prior-art CoDeSys 1.5 (1997) and WAGO-I/O-PRO (1999) user's guides disclose the same functionality as described in CoDeSys 2.3 (2007).

12. **storing a program in a control device, the control device receives a message from a communications medium, the message includes instructions to suspend execution of the stored program at a particular location of the stored program;**

13. The table below cross-references limitations of this claim element to the prior art. Supporting discussion follows after the table.

Cross reference of Claim 1 elements to CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007)			
	CoDeSys 1.5 (1997)	**WAGO-I/O-PRO (1999)**	**CoDeSys 2.3 (2007)**
storing a program in a control device	download	download	download
communications medium	serial port cable or Ethernet cable	serial port cable	serial port cable or Ethernet cable
instructions to suspend execution	online stop, online breakpoints	online stop, online breakpoints	online stop, online breakpoints
at a particular location	between two cycles, at breakpoints	between two cycles, at breakpoints	between two cycles, at breakpoints

14. Zatarain (page 46) cites to the CoDeSys 2.3 (2007) manual in support of his opinion that a program created using CoDeSys 2.3 (2007) can be stored in a PLC. I note CoDeSys 1.5 (1997) and WAGO-I/O-PRO (1999) supports this same functionality. As described on page 58 of the CoDeSys 1.5 (1997) user's guide, CoDeSys 1.5 (1997) includes a method of "'Online' 'Download'" to store a program in the logic controller. WAGO-I/O-PRO (1999) includes the same "storing a program" functionality on page 6-2. The table below shows the same "storing a program" functionality was present in all.

CoDeSys 1.5 (1997) user's guide (page 58) *'Online''Download'* This command loads a compiled project to the PLC.
WAGO-I/O-PRO (1999) user's guide (page 6-2) *'Online' 'Download'* The current project is downloaded into the logic controller or the simulation environment.
CoDeSys 2.3 (2007) user's guide (page 4-70) **'Online' 'Download'** This command loads the compiled project in the PLC.
The same "storing a program" functionality was present in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007).

15. Zatarain (page 47) opines the CoDeSys 2.3 (2007) PLC receives a message from a Personal Computer (PC) "via a communications medium (e.g. using Ethernet link in testing)." CoDeSys 1.5 (1997) also supports this functionality. As described on page 57 of the user's guide, messages are passed from a PC to the logic controller via a serial interface or Ethernet link. An Ethernet link and a serial port cable are both communications mediums. WAGO-I/O-PRO (1999) includes "communications medium" functionality that is discussed on page 6-5, where it states, "The communication between PC and controller is made via a serial interface using a WAGO communication cable (Component of the IEC 1131-3 programming tool)." The table below shows the same "communications medium" functionality was present in all.

CoDeSys 1.5 (1997) user's guide (page 64)

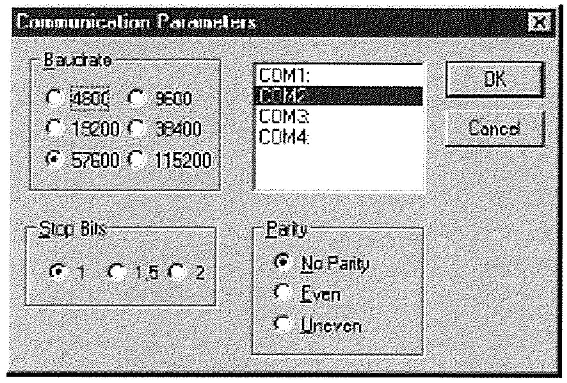


Fig. 4.23: Communication Parameters dialog

WAGO-I/O-PRO (1999) user's guide (page 6-5)

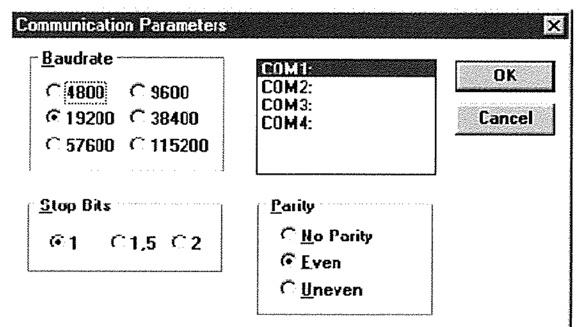
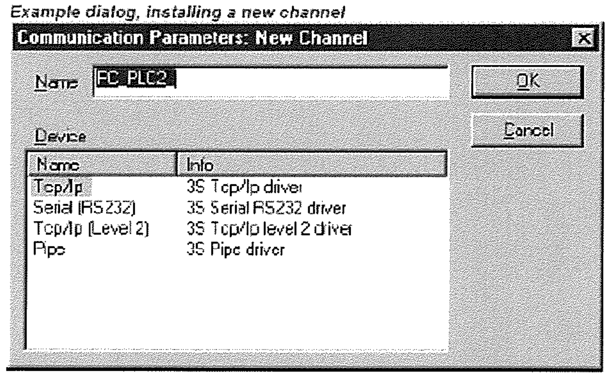


Fig. 6.2: Dialog for setting the communication parameter

CoDeSys 2.3 (2007) user's guide (page 4-79)

Example dialog, installing a new channel



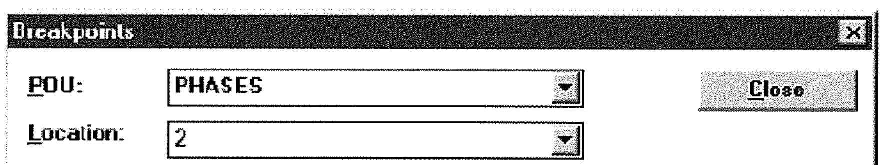
The same "communications medium" functionality was present in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007).

16. Zatarain (page 47) opines the message received by the CoDeSys 2.3 (2007) PLC "contains instructions to suspend execution of the program running on the PLC at a particular program location." CoDeSys 1.5 (1997) supports this same functionality. As described on page 58 of the user's guide, CoDeSys 1.5 (1997) includes a method of "'Online' 'Stop'" to suspend execution of the program "between two cycles." WAGO-I/O-PRO (1999) includes the same "suspend execution" functionality on page 6-2. The table below shows the same "suspend execution" functionality was present in all.

CoDeSys 1.5 (1997) user's guide (page 58) *'Online''Stop'* Stops the execution of the program loaded to the PLC.
WAGO-I/O-PRO (1999) user's guide (page 6-2) *'Online''Stop'* Stops the execution of the user program in the logic controller between two cycles.
CoDeSys 2.3 (2007) user's guide (page 4-71) 'Online' 'Stop' Symbol: Shortcut <Shift>+<F8> Stops the execution of the program in the PLC or in Simulation Mode between two cycles.
The same "suspend execution" functionality was present in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007).

17. Zatarain (page 47) opines the CoDeSys 2.3 (2007) software has an online mode that allows sending a breakpoint command to suspend execution of a stored program on a PLC at a particular location of the stored program. CoDeSys 1.5 (1997) includes this same online mode functionality. As described on page 59 of the user's guide, CoDeSys 1.5 (1997) includes a method of "'Online' 'Breakpoint Dialog'" to suspend execution of the program at a set breakpoint in the program. WAGO-I/O-PRO (1999) includes the same breakpoint "at a particular

location" functionality on page 6-17. The table below shows the same breakpoint "at a particular location" functionality was present in all.

CoDeSys 1.5 (1997) user's guide (page 59) *'Online'* *'Breakpoint Dialog'* Opens a dialog for setting and clearing breakpoints throughout the whole projects. The dialog shows all breakpoints and allows to find the position, where the breakpoint is set.
WAGO-I/O-PRO (1999) user's guide (page 6-17) *'Online'* *'Breakpoint Dialog* Instead of setting a breakpoints directly in the code and to delete same, setting can also be made in breakpoint dialog: 
CoDeSys 2.3 (2007) user's guide (page 4-72) 'Online' 'Breakpoint Dialog Box' This command opens a dialog box to edit breakpoints throughout the entire project. The dialog box also displays all breakpoints presently set.
The same breakpoint "at a particular location" functionality was present in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007).

18. **executing at least a portion of the stored program in the control device according to the instructions;**

19. Zatarain (page 49) opines the PLC in the CoDeSys 2.3 (2007) software "receives a message from a communications medium (selection of the "Run" command) to start execution of a stored program." CoDeSys 1.5 (1997) includes this same functionality. As described on page 58 of the user's guide, CoDeSys 1.5 (1997) includes a method of "'Online''Run'" that "starts the execution of the program." WAGO-I/O-PRO (1999) includes the same "executing at least a

portion" functionality on page 6-2. The table below shows the same "executing at least a portion" functionality was present in all.

CoDeSys 1.5 (1997) user's guide (page 58) *'Online''Run'* This command starts the execution of the program you successfully wrote with CoDeSys. Short Cut: <F5>.
WAGO-I/O-PRO (1999) user's guide (page 6-2) *'Online''Run'* Starts the execution of the user program in the logic controller. Short cut: <F5>
CoDeSys 2.3 (2007) user's guide (page 4-70) *'Online' 'Run'* **Symbol:** **Shortcut: <F5>** This command starts the program in the PLC or in Simulation Mode.
The same "executing at least a portion" functionality was present present in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007).

20. **suspending execution of the stored program according to the instructions;**

21. Zatarain (page 50) opines the PLC in the CoDeSys 2.3 (2007) software suspends execution of a stored program running on a PLC after executing an instruction. CoDeSys 1.5 (1997) includes this same functionality. As described on page 58 of the user's guide, CoDeSys 1.5 (1997) includes a method of "'Online''Stop'" to suspend execution of the stored program running on a PLC. WAGO-I/O-PRO (1999) includes the same "suspending execution" functionality on page 6-2. The table below shows the same "suspending execution" functionality was present in all.

CoDeSys 1.5 (1997) user's guide (page 58) *'Online''Stop'* Stops the execution of the program loaded to the PLC.
WAGO-I/O-PRO (1999) user's guide (page 6-2) *'Online''Stop'* Stops the execution of the user program in the logic controller between two cycles.
CoDeSys 2.3 (2007) user's guide (page 4-71) 'Online' 'Stop' Symbol:  Shortcut <Shift>+<F8> Stops the execution of the program in the PLC or in Simulation Mode between two cycles.
The same "suspend execution" functionality was present in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007).

22. **receiving a mode change message with instructions therein to execute the stored program in a step mode from the location in which the program was suspended.**

23. Zatarain (page 52) opines the PLC in the CoDeSys 2.3 (2007) software "receives a mode change message, e.g. the "Online" drop-down menu has several mode change options (e.g. "Single Cycle)." CoDeSys 1.5 (1997) includes this same functionality. As described on page 60 of the user's guide, CoDeSys 1.5 (1997) includes several methods for executing the stored program in step mode. The "'Online' 'Step Over'" and "'Online' 'Step In'" messages are possible examples of mode change messages. WAGO-I/O-PRO (1999) includes the same "step mode" functionality on page 6-18. The table below shows the same "step mode" functionality was present in all.

CoDeSys 1.5 (1997) user's guide (page 60)

'Online'

'Step Over'

A single step is performed. A call of POUs is treated as one step (execution is stopped after the execution of the POU). In SFC, a whole action is executed.

Short cut: <F10>.

'Online'

'Step In'

Steps to the next statement, even if it is in a different POU.

If the next statement is the call of another POU, execution stops before the first statement of the called POU.

Short cut: <F8>.

WAGO-I/O-PRO (1999) user's guide (page 6-18)

'Online'

'Step Over'

A single step is performed, whereby calling up a POU is not stopped before this step is processed. In SFC a complete action is processed.

Short cut: <F10>

'Online'

'Step In'

A single steps are processed, whereby calling up of POU's is stopped prior to executing the first POU statement.

Short cut: <F8>

CoDeSys 2.3 (2007) user's guide (page 4-72)

'Online' 'Step in'

Shortcut: <F8>

A single step is executed. The program is stopped before the first instruction of a called POU.

'Online' 'Step over'

Symbol:  **Shortcut: <F10>**

This command causes a single step to execute. If a POU is called, the program stops after its execution. In SFC a complete action is executed.

The same "step mode" functionality was present in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007).

Analysis Claim 2

24. **Claim 2 - The method of claim 1, further comprising repeating executing the at least a portion of the stored program and suspending execution of the stored program, in response to another message from the communications medium.**

25. Zatarain (page 54) opines the execution of a stored program in the PLC of the CoDeSys 2.3 (2007) software "can be repeated and the suspension of execution of the stored program can be repeated in response to another message from the communications medium (i.e. 'single cycle')." CoDeSys 1.5 (1997) includes this same functionality. As described on page 60 of the user's guide, CoDeSys 1.5 (1997) includes a method called "'Online' 'Single Cycle'" for executing a single cycle of the program and then suspending execution. Subsequent "'Online' 'Single Cycle'" messages repeat the execution of a single cycle of the program. WAGO-I/O-PRO (1999) includes the same "repeating executing" functionality on page 6-18. The table below shows the same "repeating executing" functionality was present in all.

CoDeSys 1.5 (1997) user's guide (page 60)

'Online''Single Cycle'

Execution starts and stops at the end of one PLC cycle (or when reaching the next breakpoint).

WAGO-I/O-PRO (1999) user's guide (page 6-18)

'Online'

'Single Cycle'

The single cycle is activated both in simulation mode as well as in operation with a programmable fieldbus controller.

Using this function a logic controller cycle is processed and then stopped.

Short cut: <Ctrl>+<F5>

CoDeSys 2.3 (2007) user's guide (page 4-73)

'Online' 'Single Cycle'

Shortcut: <Ctrl>+<F5>

This command executes a single PLC Cycle and stops after this cycle.

This command can be repeated continuously in order to proceed in single cycles.

The Single Cycle ends when the 'Online' 'Run' command is executed.

The same "repeating execution" functionality was present in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007).

<u>Analysis Claim 3</u>

26. **Claim 3 - The method of claim 1, further comprising providing data to the communications medium in response to a data request message from a network while execution of the stored program is suspended.**

27. Zatarain (page 56) opines the PLC in the CoDeSys 2.3 (2007) software provides "data to the communications medium in response to a data request message from a network while execution of the stored program is suspended." CoDeSys 1.5 (1997) includes this same functionality. As described on page 119 of the user's guide, CoDeSys 1.5 (1997) includes a method called "'Extras' 'Read Trace'" to read parameters out of the logic controller. The manual

further discloses the message causes a "read" of the logic controller's trace buffer. WAGO-I/O-PRO (1999) includes the same "providing data" functionality on page 6-11. The table below shows the same "providing data" functionality was present in all.

CoDeSys 1.5 (1997) user's guide (page 119) *'Extras' 'Read Trace'* This command uploads the current trace buffer. The values of the variables are displayed.
WAGO-I/O-PRO (1999) user's guide (page 6-11) *'Extras' 'Read Trace'* The current trace buffer is read out of the logic controller and the values of the selected variables is displayed.
CoDeSys 2.3 (2007) user's guide (page 6-63) **'Extra' 'Read Trace'** **Symbol:** With this command the present trace buffer is read from the PLC, and the values of the selected variables are displayed.
The same "providing data" functionality was present in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007).

<u>Analysis Claim 5</u>

28. **Claim 5 - The method of claim 1, wherein the mode change message further comprises a step type, and wherein executing the at least a portion of the stored program and suspending execution of the stored program are done according to the step type.**

29. Zatarain (page 59) opines the PLC in the CoDeSys 2.3 (2007) software "receives a mode change message, e.g. the "Online" drop-down menu has several mode change options (e.g., "Step In")." CoDeSys 1.5 (1997) includes this same functionality. As described on page 60 of the user's guide, CoDeSys 1.5 (1997) includes methods called "'Online' 'Step Over'" and "'Online' 'Step In.'" These are two step types that execute and then suspend execution of the

stored program. WAGO-I/O-PRO (1999) includes the same “step type” functionality on page 6-18. The table below shows the same “step type” functionality was present in all.

CoDeSys 1.5 (1997) user’s guide (page 60) *'Online'* *'Step Over'* A single step is performed. A call of POUs is treated as one step (execution is stopped after the execution of the POU). In SFC, a whole action is executed. Short cut: <F10>. *'Online'* *'Step In'* Steps to the next statement, even if it is in a different POU. If the next statement is the call of another POU, execution stops before the first statement of the called POU. Short cut: <F8>.
WAGO-I/O-PRO (1999) user’s guide (page 6-18) *'Online'* *'Step Over'* A single step is performed, whereby calling up a POU is not stopped before this step is processed. In SFC a complete action is processed. Short cut: <F10> *'Online'* *'Step In'* A single steps are processed, whereby calling up of POU's is stopped prior to executing the first POU statement. Short cut: <F8>
CoDeSys 2.3 (2007) user’s guide (page 4-72) **'Online' 'Step in'** **Shortcut: <F8>** A single step is executed. The program is stopped before the first instruction of a called POU. **'Online' 'Step over'** **Symbol:**  **Shortcut: <F10>** This command causes a single step to execute. If a POU is called, the program stops after its execution. In SFC a complete action is executed.
The same “step mode” functionality was present in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007).

Analysis Claim 10

30. **Claim 10 - The method of claim 5, further comprising repeating executing the at least a portion of the stored program and suspending execution of the stored program, in response to another message from the communications medium.**

31. Zatarain (page 63) opines that the execution of a stored program in the PLC of the CoDeSys 2.3 (2007) software "can be repeated and the suspension of execution of the stored program can be repeated in response to another message from the communications medium (i.e. "single cycle")." CoDeSys 1.5 (1997) includes this same functionality. As previously discussed, CoDeSys 1.5 (1997) supports repeating execution of a program cycle by repeating the "'Online' 'Single Cycle'" message. WAGO-I/O-PRO (1999) includes the same "repeating execution" functionality on page 6-18. The table below shows the same "repeating execution" functionality was present in all.

CoDeSys 1.5 (1997) user's guide (page 60) *'Online''Single Cycle'* Execution starts and stops at the end of one PLC cycle (or when reaching the next breakpoint).
WAGO-I/O-PRO (1999) user's guide (page 6-18) *'Online'* *'Single Cycle'* The single cycle is activated both in simulation mode as well as in operation with a programmable fieldbus controller. Using this function a logic controller cycle is processed and then stopped. Short cut: <Ctrl>+<F5>
CoDeSys 2.3 (2007) user's guide (page 4-73) 'Online' 'Single Cycle' Shortcut: <Ctrl>+<F5> This command executes a single PLC Cycle and stops after this cycle. This command can be repeated continuously in order to proceed in single cycles. The Single Cycle ends when the 'Online' 'Run' command is executed.
The same "repeating execution" functionality was present in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007).

<u>Analysis Claim 11</u>

32. **Claim 11 - The method of claim 5, further comprising providing data to the communications medium in response to a data request message from the network while execution of the stored program is suspended.**

33. Zatarain (page 65) opines the PLC "provides PLC memory data to the communications medium in response to a data request message from the online access tool (i.e. WAGO-I/O-PRO CAA) over an Ethernet link while execution of the stored program is suspended." CoDeSys 1.5 (1997) includes this same functionality. As previously discussed, CoDeSys 1.5 (1997) includes a method called "'Extras' 'Read Trace'" to read parameters out of the logic controller. WAGO-

I/O-PRO (1999) includes the same "providing data" functionality on page 6-11. The table below shows the same "providing data" functionality was present in all.

CoDeSys 1.5 (1997) user's guide (page 119) *'Extras' 'Read Trace'* This command uploads the current trace buffer. The values of the variables are displayed.
WAGO-I/O-PRO (1999) user's guide (page 6-11) *'Extras' 'Read Trace'* The current trace buffer is read out of the logic controller and the values of the selected variables is displayed.
CoDeSys 2.3 (2007) user's guide (page 6-63) **'Extra' 'Read Trace'** **Symbol:**  With this command the present trace buffer is read from the PLC, and the values of the selected variables are displayed.
The same "providing data" functionality was present in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007).

<u>Analysis Claim 14</u>

34. The table below shows both CoDeSys 1.5 (1997) and WAGO-I/O-PRO (1999) include the same functionality cited by Zatarain from CoDeSys 2.3 (2007) in his analysis of Claim 14 of the '232 patent.

Cross reference of Claim 14 elements to CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007)			
	CoDeSys 1.5 (1997)	**WAGO-I/O-PRO (1999)**	**CoDeSys 2.3 (2007)**
A method for performing a function in a control device…	X	X	X
providing a control device …	X	X	X
receiving a mode change message …	X	X	X
suspending execution of the program…	X	X	X
receiving a step command message …	X	X	X
executing at least a portion of the program …	X	X	X
suspending execution of the program …	X	X	X

35. **Claim 14 - A method of performing a function in a control device comprising:**

36. As discussed, CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007) disclose such methods for performing a function in a control device.

37. **providing a control device that selectively executes a program and receives messages from a network;**

38. As discussed, CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007) disclose this functionality.

39. **receiving a mode change message from the network;**

40. As discussed, CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007) disclose this functionality.

41. **suspending execution of the program according to the mode change message;**

42. As discussed, CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007) disclose this functionality.

43. **receiving a step command message from the network;**

44. As discussed, CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007) disclose this functionality.

45. **executing at least a portion of the program in the control device according to the message;**

46. As discussed, CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007) disclose this functionality.

47. **suspending execution of the program according to the message.**

48. As discussed, CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007) disclose this functionality.

Count Three: '813 Patent

49. I understand Plaintiffs are asserting Claims 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21 and 22 of U.S. Patent No. 6,801,813 (the '813 patent).

50. Zatarain (page 110) opines: "I do not believe any of the claim terms in the '813 patent require explanation or a Court's interpretation. The terms are clear and easily understood not only by a PHOSITA, but simply by laypeople as well." This opinion is not correct. A number of terms used in this patent would not be clear and easily understood by a layperson. "Execution engine," "ladder logic," and "interpret code" are examples of terms which would not be easily understood by a layperson and which are relevant to my opinion in this analysis (but not the only examples of terms in the '813 patent requiring interpretation before a layperson could understand them).

Analysis Claim 1

51. **Claim 1 - An industrial controller system comprising:**

52. While some WAGO-I/O systems may be an industrial controller system, as discussed below, it does not include all of the elements of this claim.

53. **a file system residing in a program memory of an industrial controller, the file system having a plurality of file system services;**

54. While some WAGO-I/O systems may include a file system, as discussed below, the file system is not provided in the context of the elements of this claim.

55. **an execution engine residing in the program memory of the industrial controller, the execution engine adapted to interpret code from an industrial control program, the industrial control program including at least one instruction utilizing one or more of the plurality of file system services.**

56. Zatarain (pages 115-116) cites generally and without specifics to the Albers 30(b)(6) deposition (Albers) to support the contention that the WAGO-I/O system practices this claim element. This is not correct. In fact, as discussed below, the Albers deposition shows the WAGO-I/O system DOES NOT practice this claim element. Specifically, as detailed further below, the WAGO-I/O system does not include an execution engine on the industrial controller

adapted to interpret code, because the WAGO-I/O systems support compiled computer programming languages and not interpreted languages.

57. Consider specifically the following testimony by Albers, "The customer writes a program into an editor; he compiles the written program, and, thus, creates an executable machine code. Then he loads the executable machine code onto the controller." (Page 78, lines 17-21) This testimony by Albers corresponds with my analysis that programs in the WAGO-I/O system are compiled rather than interpreted.

58. As further evidence that the WAGO-I/O system does not include an execution engine on the industrial controller adapted to interpret code, consider the following testimony by Albers, "Q When a customer uses the PRO CAA software to write a program in Ladder Logic instruction, before that program is loaded on to the controller, does the program have to be compiled by the customer on the PC? A Yes, it has to be compiled before." (Page 85, line 21 – Page 86, line 1) Again, this testimony by Albers corresponds with my analysis.

59. Before loading a compiled program on a computer, compiled programs undergo an intermediate step that processes the human-written program to create executable machine code, typically ones and zeros. This process of translating the human-written programming language, such as ladder logic, into executable machine code is called compiling. People use editors to write computer programs, which must be transformed by a compiler before the program can be executed by the controller.

60. In contrast, interpreted programs do not undergo the compiling process before being loaded on a computer. Rather, interpreted programs require an interpreter to process the human-written programming language. An interpreter transforms and executes a human-written computer program into executable machine code one instruction at a time.

61. For a further discussion of this subject see the textbook, Programming Industrial Control Systems Using IEC 11131-3 by R.W. Lewis, pages 169 through 184, which is incorporated by reference.

62. The language of the patent is clear this claim element requires a program interpreter. The words "interpret," "interprets," or "interpretable" are used more than ten times in the specification. One of ordinary skill in the art understands that interpreting code has a very

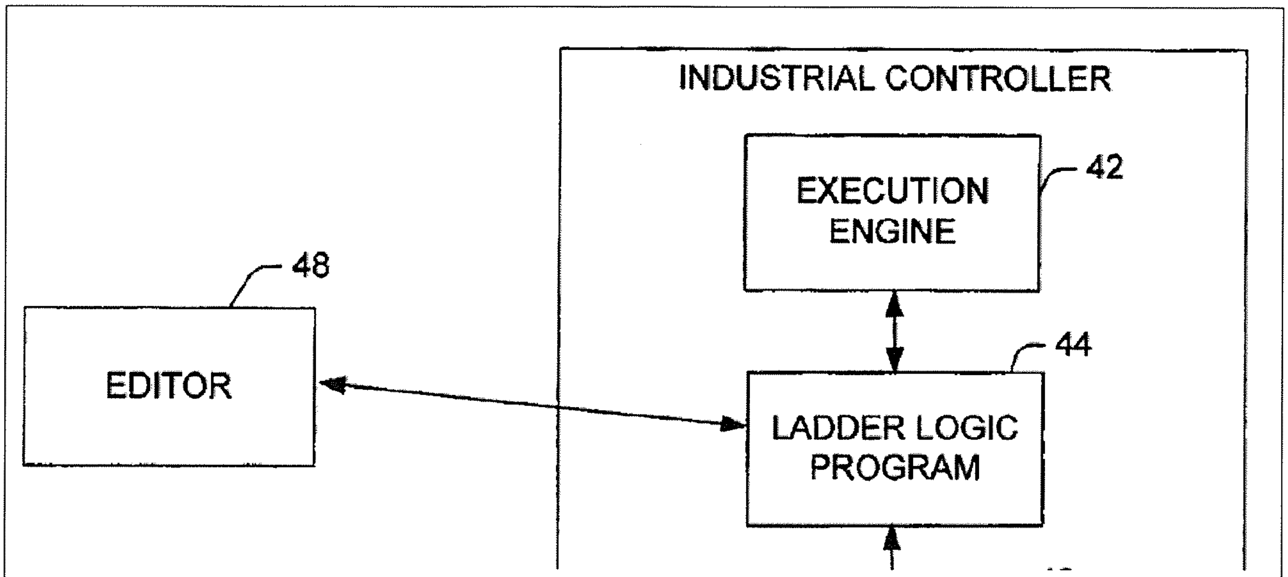
specific meaning that requires a program interpreter. The word "compile" does not appear in the patent.

63. Consider the following from the '813 patent: "Once the ladder logic program is complete, the PC software converts the graphs into the corresponding ladder logic commands. The ladder logic command are then transferred to the PLC and stored in the PLC memory." (Col. 2, lines 1-4) A person of ordinary skill in the art understands ladder logic commands include words such as: "Add," "Subtract," "Multiply," "Divide," "Count Up," "Count Down," and the like. The ONLY way the execution engine of the '813 patent could execute these commands is if it includes a program interpreter or something equivalent.

64. As further evidence that the execution engine of the patent includes a program interpreter, consider the following from the specification: "The current program running on the industrial controller can only be selected and/or changed by the editor." (Col. 2, lines 10-12) One of ordinary skill in the art understands that only interpreted programs can be edited and changed while they are running.

65. As still further evidence the execution engine of the patent includes a program interpreter, consider the following from the specification: "An execution engine is adapted to interpret new instructions that invoke the services of the file system." (Col. 2, lines 40-42) One of ordinary skill in the art understands that instructions are words such as: "Load," "Store," "Set," "Reset," and the like. The ONLY way the execution engine of the '813 patent could execute these instructions is if it includes a program interpreter or something equivalent. The textbook, <u>Programming Industrial Control Systems Using IEC 11131-3</u> by R.W. Lewis, supports this assertion as described on pages 169 through 184, which is incorporated by reference.

66. The illustration below from Figure 2 of the patent shows that an editor is interacting with the ladder logic program. As discussed, people use editors to write computer programs. The only way these programs could be directly executed by the execution engine is if the execution engine includes a program interpreter or something equivalent.



Note the ladder logic program is being executed directly by the execution engine. The only way the execution engine can execute the ladder logic program is if the execution engine includes a program interpreter or something equivalent.

67. Finally, I note that EVERY time the specification of the patent uses the words interpret, interprets or interpretable it is in the context of interpreting instructions. As discussed, program instructions are human-written words and the ONLY way the execution engine of the '813 patent could execute these instructions is if it includes a program interpreter or something equivalent.

68. The WAGO-I/O system does not include an interpreter. I base this conclusion on careful review of the WAGO-I/O system and its manuals. Furthermore, Zatarain does not contend the WAGO-I/O system includes an interpreter.

69. There is an abundance of evidence the WAGO-I/O system supports compiled programs rather than interpreted languages. For example, the CoDeSys 2.3 manual uses the word "compile" in this context more than 50 times. (Page 1-1, page 3-9, page 3-13, page 4-6, page 4-11, page 4-13, page 4-14, page 4-17, etc.) Furthermore, there is an entire section in the manual devoted to "Target Settings." CoDeSys needs to know the "Target Settings" so that it can compile the program and create executable code specifically for the target PLC. There is no content in the manuals at all that discusses program interpreters.

Analysis Claim 2

70. **Claim 2 - The system of claim 1, the file system and the execution engine being adapted to load user defined routine files upon loading an industrial control program having one or more header instructions for including a user defined routine file, the included user defined routine file being loaded into the same program space as the industrial control program.**

71. As discussed, the WAGO-I/O system does not practice Claim 1 of the '813 patent and thus does not practice this dependent claim.

72. Furthermore, there is no notion in the WAGO-I/O system of loading user defined routine files upon loading an industrial control program. As described in Section 12 of the technical manuals for the WAGO 758-870 IPC controllers (and in Exhibit C to Zatarain, page 9), user defined routine files are associated with industrial control programs during the compiling process, which must be done before executing an industrial control program on a controller. As discussed, the '813 patent does not disclose compiling.

Analysis Claim 3

73. **Claim 3 - The system of claim 2, the user defined routine files being stored in a memory device separate from the program memory.**

74. As discussed, the WAGO-I/O system does not practice Claim 2 of the '813 patent and thus does not practice this dependent claim. To the extent user defined routine files may be stored in a memory device separate from the program memory, the prior-art WAGO-I/O-PRO (1999) product discloses this functionality as described on pages 1-4, 3-3 and 3-73.

75. It is my understanding that WAGO-I/O-PRO (1999) precedes the July 30, 2001 filing date of the '813 patent.

Analysis Claim 4

76. **Claim 4 - The system of claim 3, the memory device being located at one of the industrial controller and a remote location from the industrial controller.**

77. As discussed, the WAGO-I/O system does not practice Claim 3 of the '813 patent and thus does not practice this dependent claim. To the extent a memory device may be located at

one of the industrial controller and a remote location from the industrial controller, the prior-art WAGO-I/O-PRO (1999) product discloses this functionality as described on pages 1-4, 3-73, 6-5 and 7-6.

Analysis Claim 5

78. **Claim 5 - The system of claim 1, the file system and the execution engine being adapted to load one or more recipe files into an executing industrial control program upon executing a load instruction in an industrial control program.**

79. As discussed, the WAGO-I/O system does not practice Claim 1 of the '813 patent and thus does not practice this dependent claim.

80. Furthermore, I have found no evidence the WAGO-I/O system includes a load instruction for loading recipe files (and Zatarain does not point specifically to one). The word "recipe" does not even appear in the CoDeSys 2.3 (2007) manual.

Analysis Claim 6

81. **Claim 6 - The system of claim 5, the recipe files being stored at a memory device separate from the program memory.**

82. As discussed, the WAGO-I/O system does not meet the limitations of Claim 5 and thus does not meet the limitations of this dependent claim. To the extent files may be stored at a memory device separate from the program memory, the prior-art WAGO-I/O-PRO (1999) product discloses this functionality as described on pages 1-4, 3-3 and 3-73.

Analysis Claim 7

83. **Claim 7 - The system of claim 6, the memory device being located at one of the industrial controller and a remote location from the industrial controller.**

84. As discussed, the WAGO-I/O system does not meet the limitations of Claim 6 and thus does not meet the limitations of this dependent claim. To the extent a memory device may be located at one of the industrial controller and a remote location from the industrial controller, the prior-art WAGO-I/O-PRO (1999) product discloses this functionality as described on pages 1-4, 3-73, 6-5 and 7-6.

Analysis Claim 10

85. **Claim 10 - The system of claim 1, the file system and the execution engine being adapted to log measured data into a file upon executing an instruction in an industrial control program to record the measured data.**

86. As discussed, the WAGO-I/O system does not meet the limitations of Claim 1 and thus does not meet the limitations of this dependent claim. Furthermore, the handling of measured data in current WAGO-I/O systems is the same as the handling of measured data in the prior-art WAGO-I/O-PRO (1999) product as described on pages 6-7 through 6-9.

Analysis Claim 11

87. **Claim 11 - The system of claim 10, the file system and the execution engine being adapted to retrieve measured data from a file upon executing an instruction in an industrial control program to load the measured data.**

88. As discussed, the WAGO-I/O system does not meet the limitations of Claim 10 and thus does not meet the limitations of this dependent claim. Furthermore, the handling of measured data in current WAGO-I/O systems is the same as the handling of measured data in the prior-art WAGO-I/O-PRO (1999) product as described on pages 6-7 through 6-9.

Analysis Claim 12

89. **Claim 12 - The system of claim 10, the measured data file being stored at a memory device separate from the program memory.**

90. As discussed, the WAGO-I/O system does not meet the limitations of Claim 10 and thus does not meet the limitations of this dependent claim. To the extent measured data file may be stored at a memory device separate from the program memory, the prior-art WAGO-I/O-PRO (1999) product discloses this functionality as described on pages 1-4, 3-3 and 3-73.

Analysis Claim 13

91. **Claim 13 - The system of claim 12, the memory device being located at one of the industrial controller and a remote location from the industrial controller.**

92. As discussed, the WAGO-I/O system does not meet the limitations of Claim 12 and thus does not meet the limitations of this dependent claim. To the extent a memory device may be

located at one of the industrial controller and a remote location from the industrial controller, the prior-art WAGO-I/O-PRO (1999) product discloses this functionality as described on pages 1-4, 3-73, 6-5 and 7-6.

Analysis Claim 14

93. **Claim 14 - The system of claim 1, the file system and the execution engine being adapted to log trend data into a file upon executing an instruction in an industrial control program to record the trend data.**

94. As discussed, the WAGO-I/O system does not meet the limitations of Claim 1 and thus does not meet the limitations of this dependent claim. Furthermore, the handling of trend data in current WAGO-I/O systems is the same as the handling of trend data in the prior-art WAGO-I/O-PRO (1999) product as described on pages 6-11 and 6-13.

Analysis Claim 15

95. **Claim 15 - The system of claim 14, the file system and the execution engine being adapted to retrieve trend data from a file upon executing an instruction in an industrial control program to load the trend data.**

96. As discussed, the WAGO-I/O system does not meet the limitations of Claim 14 and thus does not meet the limitations of this dependent claim. Furthermore, the handling of trend data in current WAGO-I/O systems is the same as the handling of trend data in the prior-art WAGO-I/O-PRO (1999) product as described on pages 6-11 and 6-13.

Analysis Claim 16

97. **Claim 16 - The system of claim 14, the trend data file being stored at a memory device separate from the program memory.**

98. As discussed, the WAGO-I/O system does not meet the limitations of Claim 14 and thus does not meet the limitations of this dependent claim. To the extent a trend data file may be stored at a memory device separate from the program memory, the prior-art WAGO-I/O-PRO (1999) product discloses this functionality as described on pages 1-4, 3-3 and 3-73.

Analysis Claim 17

99. **Claim 17 - The system of claim 16, the memory device being located at one of the industrial controller and a remote location from the industrial controller.**

100. As discussed, the WAGO-I/O system does not meet the limitations of Claim 16 and thus does not meet the limitations of this dependent claim. To the extent a memory device may be located at one of the industrial controller and a remote location from the industrial controller, the prior-art WAGO-I/O-PRO (1999) product discloses this functionality as described on pages 1-4, 3-73, 6-5 and 7-6.

Analysis Claim 20

101. **Claim 20 - The system of claim 1, the industrial control program being a ladder logic program.**

102. As discussed, the WAGO-I/O system does not meet the limitations of Claim 1 and thus does not meet the limitations of this dependent claim. To the extent an industrial control program may be a ladder logic program, the prior-art WAGO-I/O-PRO (1999) product discloses this functionality as described on page 2-26.

Analysis Claim 21

103. **Claim 21 - A method for providing, an industrial controller with the functionality associated with utilizing a file system residing in the industrial controller, the method comprising:**

104. While the WAGO-I/O system may associate file system functionality with an industrial controller, as discussed below, it does not include all of the elements of this claim.

105. **developing a file system and loading the file system on an industrial controller, the file system having a plurality of file system services;**

106. While some WAGO-I/O systems may include a file system, as discussed below, the file system is not provided in the context of the elements of this claim.

107. **developing an execution engine that interprets instructions of an industrial control program that utilizes at least one of the plurality of file system services.**

108. As discussed, the WAGO-I/O system does not interpret instructions and thus does not meet the limitations of this claim element.

Analysis Claim 22

109. **Claim 22 - The method of claim 21, further comprising developing an industrial control program including at least one instruction that utilizes one or more file system services and downloading the industrial control program to the industrial controller.**

110. As discussed, the WAGO-I/O system does not meet the limitations of Claim 21 and thus does not meet the limitations of this dependent claim. As for developing an industrial control program including at least one instruction that utilizes one or more file system services and downloading the industrial control program to the industrial controller, the prior-art WAGO-I/O-PRO (1999) product discloses this functionality as described on pages 3-29 and 6-2.

Count Five: '415 Patent

111. I understand Plaintiffs are asserting Claims 1, 2, 3, 4, 5, 8 and 13 of U.S. Patent No. 7,065,415 (the '415 patent). Zatarain addresses Claims 1, 2, 3, 4, 5 and 8. I will also address these claims and Claim 13. In the event Plaintiffs assert additional contentions to this patent, I reserve the opportunity to address these additional contentions in a supplemental report.

112. Zatarain (page 147) opines "I do not believe any of the claim terms in the '415 patent require explanation or a Court's interpretation, other than the term "recipe file," as the remainder of the terms are clear and easily understood not only by a PHOSITA, but simply by laypeople as well." This opinion is not correct. A number of terms used in this patent would not be clear and easily understood by a layperson. "Execution engine," "ladder logic," and "implementation for converting" are examples of terms which would not be easily understood by a layperson and which are relevant to my opinion in this analysis (but not the only examples of terms in the '415 patent requiring interpretation before a layperson could understand them).

113. Zatarain (pages 151-152) cites generally and without specifics to the Albers 30(b)(6) deposition to support the contention that the WAGO-I/O system infringes the '415 patent. This contention is not correct. The Albers deposition does not support this contention and as detailed below, the WAGO-I/O system does not practice each element of each asserted claim.

114. Furthermore the ladder logic functionality in the WAGO-I/O system generally cited to in the Albers deposition was present in the prior-art CoDeSys 1.5 (1997) and WAGO-I/O-PRO (1999). This is shown below via examples from the CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007) manuals that describe the functionality of interest in the WAGO-I/O system.

115. It is my understanding that CoDeSys 1.5 (1997) and WAGO-I/O-PRO (1999) precede the August 23, 2004 filing date of the '415 patent and the July 30, 2001 filing date of the '813 patent.

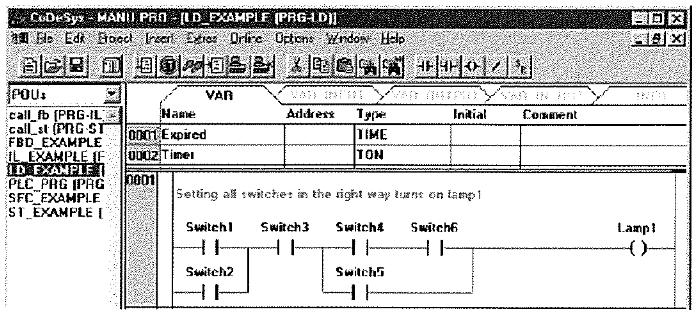
Comparison of Ladder Logic Functionality in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007)

116. The table below shows the ladder logic editor in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007) had the same functions in their respective user interfaces.

CoDeSys 1.5 (1997) user's guide (page 98)

5.3.2 The Ladder Diagram editor

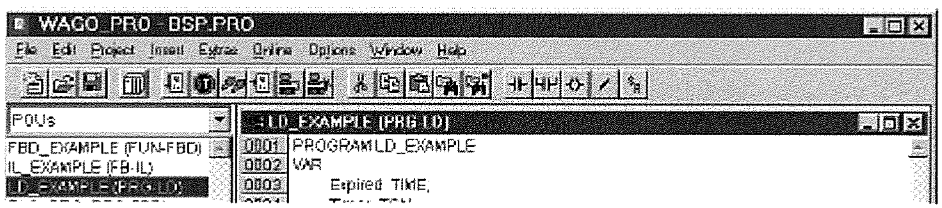
A POU written in LD:



WAGO-I/O-PRO (1999) user's guide (page 3-54)

3.3.3.2 Ladder diagram editor

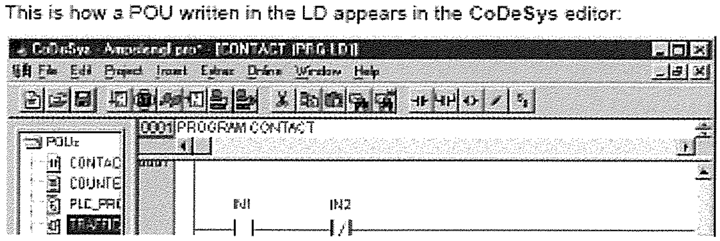
A POU written in LD in the WAGO-I/O-*PRO* editor appears as follows:



CoDeSys 2.3 (2007) user's guide (page 5-33)

5.4.3 The Ladder Editor

This is how a POU written in the LD appears in the CoDeSys editor:



The same ladder logic functionality was present in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007).

117. The table below shows both CoDeSys 1.5 (1997) and WAGO-I/O-PRO (1999) had the same ladder logic functions found by Zatarain to be present in CoDeSys 2.3 (2007). The excerpts are taken from the respective user's guides.

CoDeSys 1.5 (1997) ladder logic functions (pages 99-102)	**WAGO-I/O-PRO (1999) ladder logic functions** (pages 3-55 – 3-58)	**CoDeSys 2.3 (2007) ladder logic functions** (pages 5-35 – 5-38)
'Insert' 'Contact'	'Insert' 'Contact'	'Insert' 'Contact'
'Insert' 'Parallel Contact'	'Insert' 'Parallel Contact'	'Insert' 'Parallel Contact'
'Insert' 'Function Block'	'Insert' 'Function Block'	'Insert' 'Function Block'
'Insert' 'Coil'	'Insert' 'Coil'	'Insert' 'Coil'
'Insert' 'Operator with EN'	'Insert' 'Box with EN'	'Insert' 'Box with EN'
'Insert' 'Function Block with EN'	'Insert' 'Function Block with EN'	'Insert' 'Function Block with EN'
'Insert' 'Function with EN'	'Insert' 'Function with EN'	'Insert' 'Function with EN'
'Insert' 'Insert at Blocks'	'Insert' 'Insert at blocks'	'Insert' 'Insert at blocks'
'Insert' 'Jump'	'Insert' 'Jump'	'Insert' 'Jump'
'Insert' 'Return'	'Insert' 'Return'	'Insert' 'Return'
'Extras' 'Paste after'	'Extras' 'Paste after'	'Extras' 'Paste after'
'Extras' 'Paste below'	'Extras' 'Paste below'	'Extras' 'Paste below'
'Extras' 'Paste above'	'Extras' 'Paste above'	'Extras' 'Paste above'
'Extras' 'Negate'	'Extras' 'Negate'	'Extras' 'Negate'
'Extras''Set/Reset'	'Extras' 'Set/Reset'	'Extras' 'Set/Reset'
The same ladder logic functionality was present in CoDeSys 1.5 (1997), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007).		

<u>Analysis Claim 1</u>

118. **Claim 1 - An editor for developing ladder logic programs that control operation of an industrial controller system, the editor comprising:**

119. While the WAGO-I/O system may include an editor for ladder logic program, as discussed below, it does not include each of the elements of this claim.

120. **a first instruction that employs a file system that resides on an industrial controller to log data to a file containing ladder logic instructions;**

121. Zatarain cites to nothing that supports the position that the WAGO-I/O system includes the functionality of this claim element. In the event Plaintiffs do point to any specific functionality in the future to support their position with respect to this claim element, I reserve the opportunity to analyze their position in a supplemental report.

122. I have found no evidence the ladder logic editor supports logging data to a file containing ladder logic instructions, and, after careful review of the entire CoDeSys 2.3 (2007) manual and my testing of the WAGO-I/O system products, I conclude that it does not.

123. **a second instruction that employs the file system to retrieve the data from the file containing ladder logic instructions;**

124. Zatarain cites to nothing that supports the position that the WAGO-I/O system includes the functionality of this claim element. In the event Plaintiffs do point to any specific functionality in the future to support their position with respect to this claim element, I reserve the opportunity to analyze their position in a supplemental report.

125. I have found no evidence the ladder logic editor supports retrieving data from a file containing ladder logic instructions, and, after careful review of the entire CoDeSys 2.3 (2007) manual and my testing of the WAGO-I/O system products, I conclude that it does not.

126. **an implementation for converting the ladder logic instructions into instructions understandable and executable by an execution engine in the industrial controller.**

127. As detailed with regards to the '813 patent, the WAGO-I/O system does not include the execution engine of the '415 patent and thus does not meet the limitations of this claim element.

<u>Analysis Claim 2</u>

128. **Claim 2 - The editor of claim 1, wherein the data logged to the file and retrieved from the file is measured data.**

129. Zatarain cites to nothing that supports the position that the WAGO-I/O system includes the functionality of this claim element. In the event Plaintiffs do point to any specific functionality in the future to support their position with respect to this claim element, I reserve the opportunity to analyze their position in a supplemental report.

130. I have found no evidence the ladder logic editor supports logging or retrieving measured data from a file containing ladder logic instructions, and, after careful review of the entire CoDeSys 2.3 (2007) manual and my testing of the WAGO-I/O system products, I conclude that it does not.

Analysis Claim 3

131. **Claim 3 - The editor of claim 1, wherein the data logged to the file and retrieved from the file is trend data.**

132. Zatarain cites to nothing that supports the position that the WAGO-I/O system includes the functionality of this claim element. In the event Plaintiffs do point to any specific functionality in the future to support their position with respect to this claim element, I reserve the opportunity to analyze their position in a supplemental report.

133. I have found no evidence the WAGO-I/O editor supports logging or retrieving trend data from a file containing ladder logic instructions, and, after careful review of the entire CoDeSys 2.3 (2007) manual and my testing of the WAGO-I/O system products, I conclude that it does not.

Analysis Claim 4

134. **Claim 4 - The editor of claim 1, wherein the data logged to the file and retrieved from the file is a recipe file.**

135. Zatarain cites to nothing that supports the position that the WAGO-I/O system includes the functionality of this claim element. In the event Plaintiffs do point to any specific functionality in the future to support their position with respect to this claim element, I reserve the opportunity to analyze their position in a supplemental report.

136. I have found no evidence the WAGO-I/O editor supports logging or retrieving a recipe file from a file containing ladder logic instructions, and, after careful review of the entire CoDeSys 2.3 (2007) manual and my testing of the WAGO-I/O system products, I conclude that it does not.

Analysis Claim 5

137. **Claim 5 - The editor of claim 1, wherein the data retrieved from the file is a user defined routine file.**

138. Zatarain cites to nothing that supports the position that the WAGO-I/O system includes the functionality of this claim element. In the event Plaintiffs do point to any specific functionality in the future to support their position with respect to this claim element, I reserve the opportunity to analyze their position in a supplemental report.

139. I have found no evidence the WAGO-I/O editor supports retrieving a user defined routing file from a file containing ladder logic instructions, and, after careful review of the entire CoDeSys 2.3 (2007) manual and my testing of the WAGO-I/O system products, I conclude that it does not.

Analysis Claim 8

140. **Claim 8 - The editor of claim 1, further comprising a plurality of additional instructions that facilitate utilizing file system services of the file system.**

141. Zatarain cites to nothing that supports the position that the WAGO-I/O system includes the functionality of this claim element. In the event Plaintiffs do point to any specific functionality in the future to support their position with respect to this claim element, I reserve the opportunity to analyze their position in a supplemental report.

142. As discussed, the WAGO-I/O system does not meet the limitations of Claim 1 and thus does not meet the limitations of this dependent claim.

Analysis Claim 13

143. **A method for developing ladder logic programs that control operation of an industrial controller system, the method comprising: employing a file system that resides on an industrial controller to log data to a file containing ladder logic instructions; employing the file system to retrieve the data from the file containing ladder logic instructions; and converting the ladder logic instructions into instructions understandable and executable by an execution engine in the industrial controller.**

144. Zatarain does not address this claim. Nonetheless, my analysis of Claim 1 is applicable to each of the elements of this claim. In the event Plaintiffs assert contentions to this claim, I reserve the opportunity to address these contentions in a supplemental report.

Count Six: '974 Patent

145. I understand Plaintiffs are asserting Claims 1, 2, 3, 5, 6, 9, 10, 14, 15, 16, 19, 20, 24, 28, 29 and 30 of U.S. Patent No. 7,123,974 (the '974 patent). Zatarain addresses Claims 1, 2, 3, 5, 6, 9, 10, 14, 16, 24 and 29. I will also address these claims and Claim 28. In the event Plaintiffs assert additional contentions to this patent, I reserve the opportunity to address these additional contentions in a supplemental report.

146. Zatarain (page 74) opines "Other than the means plus function elements, I do not believe any of the claim terms in the '974 patent require explanation or a Court's interpretation, as the terms are clear and easily understood not only by a PHOSITA, but simply by laypeople as well." This opinion is not correct. A number of terms used in this patent would not be clear and easily understood by a layperson. "Aggregate," "tracking component," and "data field" are examples of terms which would not be easily understood by a layperson and which are relevant to my opinion in this analysis (but not the only examples of terms in the '974 patent requiring interpretation before a layperson could understand them).

Analysis Claim 1

147. **Claim 1 - An electronic audit system for an industrial control environment, comprising:**

148. Zatarain (page 78) states, "it is my opinion that the WAGO-I/O System does provide an electronic audit system for an industrial control environment," but provides no support for this opinion. This opinion is not correct. The word "audit" does not appear at all in the CoDeSys 2.3 (2007) manual. CoDeSys includes basic logging functionality. There are no functions directed towards auditing or an audit system.

149. **a recording component to log real time interactions with one or more industrial control components;**

150. Zatarain cites to the CoDeSys 2.3 (2007) manual to support the contention that CoDeSys meets this claim element. These citations point to the logging functionality of CoDeSys 2.3 (2007). As detailed below, the prior-art CoDeSys 2.2 software of 2001 (CoDeSys 2.2 (2001)) has the same logging functionality as CoDeSys 2.3 (2007).

151. It is my understanding that CoDeSys 2.2 (2001) precedes the November 19, 2002 filing date of the '974 patent.

<u>Comparison of Logging Functionality in CoDeSys 2.2 (2001) and CoDeSys 2.3 (2007)</u>

152. The table below shows the introduction to the logging functionality in the CoDeSys 2.2 (2001) and CoDeSys 2.3 (2007) manuals. Notice the text is exactly the same. CoDeSys 2.3 (2007) has the same logging functionality as CoDeSys 2.2 (2001).

CoDeSys 2.2 (2001) user's guide (page 4-75) **4.7 Log** The log stores in chronological order actions that occur during an Online session. For this purpose a binary log file (*.log) is set up. Afterward, the user can store excerpts from the appropriate project log in an external log. The log window can be opened in either Offline or Online mode and can thus serve as a direct monitor online.
CoDeSys 2.3 (2007) user's guide (page 6-18) 6.5 Log The log stores in chronological order actions that occur during an Online session. For this purpose a binary log file (*.log) is set up. Afterward, the user can store excerpts from the appropriate project log in an external log. The log window can be opened in either Offline or Online mode and can thus serve as a direct monitor online.
The same logging functionality was present in CoDeSys 2.2 (2001) and CoDeSys 2.3 (2007).

153. The table below shows the log windows in the CoDeSys 2.2 (2001) and CoDeSys 2.3 (2007) manuals. Notice they are exactly the same. CoDeSys 2.3 (2007) has the same logging functionality as CoDeSys 2.2 (2001).

CoDeSys 2.2 (2001) user's guide (page 4-76)

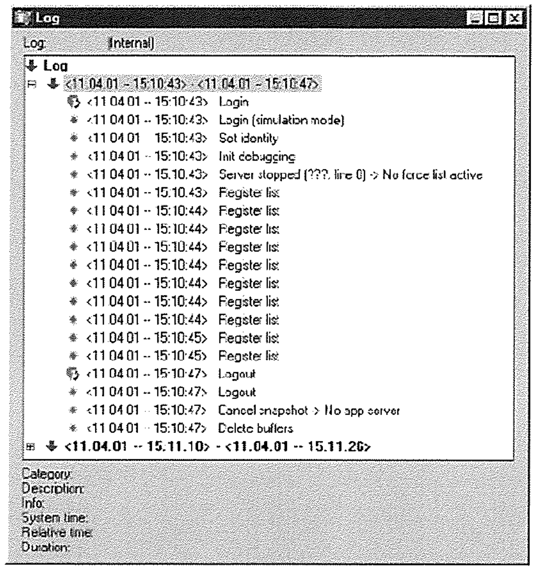
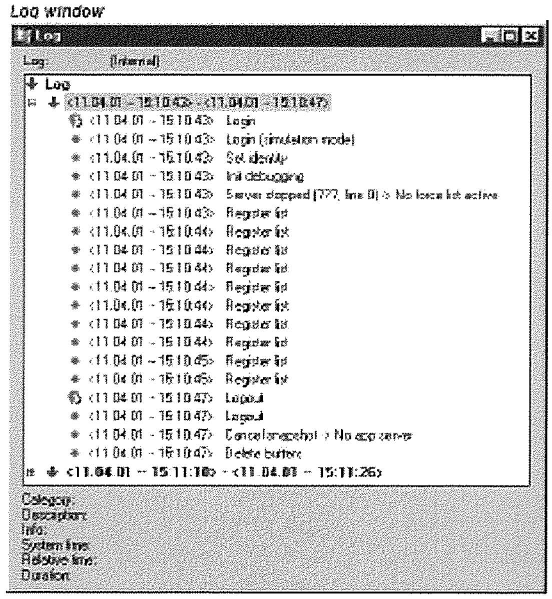


Bild 4.65 Log window

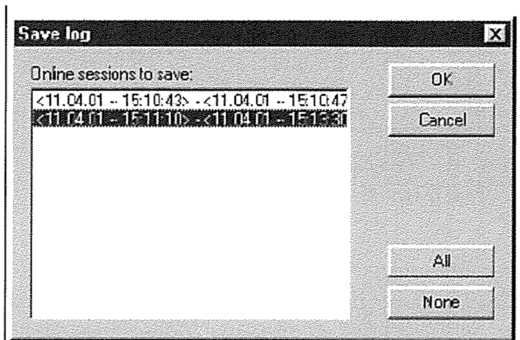
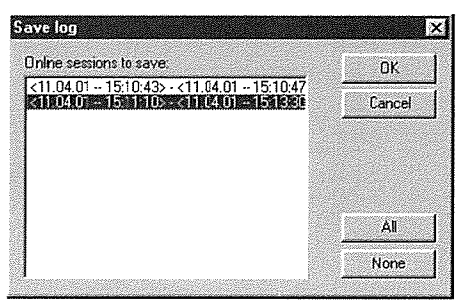
CoDeSys 2.3 (2007) user's guide (page 6-19)

Log window



The same log window was present in CoDeSys 2.2 (2001) and CoDeSys 2.3 (2007).

154. The table below shows the save log windows in the CoDeSys 2.2 (2001) and CoDeSys 2.3 (2007) manuals. Notice they are exactly the same. CoDeSys 2.3 (2007) has the same logging functionality as CoDeSys 2.2 (2001).

CoDeSys 2.2 (2001) user's guide (page 4-77)  After successful selection, the standard dialog for storing a file opens ('Save Log').
CoDeSys 2.3 (2007) user's guide (page 6-20)  After successful selection, the standard dialog for storing a file opens ('Save Log').
The same save log functionality was present in CoDeSys 2.2 (2001) and CoDeSys 2.3 (2007).

155. **a tracking component to aggregate the real time interactions to facilitate generation of audit data relating to the one or more industrial control components**

156. Zatarain again points to the same CoDeSys 2.3 (2007) logging functionality to support the contention that CoDeSys 2.3 meets this claim limitation. This is not correct. As described in the CoDeSys 2.3 (2007) manual, the logging functionality is simply the storage in "chronological order actions that occur during an Online session." (page 6-19). There is no aggregation functionality or the notion of audit data.

157. One of ordinary skill in the art would have understood that "a recording component" (first Claim 1 element) represents functionality distinct from a "tracking component" for aggregating data (second Claim 1 element). For example, Cimplicity 1997, a prior art product discussed in my previous report, contains a "Security Audit Trail" that includes two distinct functionalities. The first functionality comprises a set of standard alarms that correspond to the recording component of the first Claim 1 element. The second functionality includes a database, which corresponds to the "tracking component" that aggregates data in a database. One of ordinary skill in the art would have understood the terms "recording component" and "tracking component" in the context of products such as Cimplicity 1997 which were already known in the art at the time the '974 patent was applied for.

158. Furthermore, the language of the patent is clear that there are different and distinct functionalities for recording and tracking. For example the patent describes the following recording functionality, "the recording component records all interactions (or controlled subset of interactions) with the control system during a current session." (Col. 2, lines 49-52) This functionality is clearly distinct from the functionality in the next sentence of the patent which reads, "The respective interactions or changes are provided to the tracking component that stores the interactions in a database record associated with the control system being accessed." (Col. 2, lines 52-55)

159. CoDeSys 2.3 (2007) does not include these distinct recording and tracking functionalities. It simply includes logging functionality as was common for industrial control systems of the time.

<u>Analysis Claim 2</u>

160. **The system of claim 1, at least one of the recording component and the tracking component are associated with an access tool that interacts with the one or more industrial control components via a network.**

161. As discussed, the WAGO-I/O system does not meet the limitations of Claim 1 and thus does not meet the limitations of this dependent claim.

162. Furthermore, there is no tracking component in the WAGO-I/O system. It simply includes logging functionality as was common for industrial control systems of the time.

Analysis Claim 3

163. **The system of claim 2, the access tool includes at least one of an editing tool, a programming tool, a communications component, a monitoring component, a maintenance component, a browser, a graphical user interface (GUI), and a database application that interacts with the one or more industrial control components.**

164. As discussed, the WAGO-I/O system does not meet the limitations of Claim 2 and thus does not meet the limitations of this dependent claim.

165. To the extent an access tool may include an editing tool, a programming tool, a communications component, a monitoring component, a maintenance component and a graphical user interface (GUI) that interacts with the one or more industrial control components, the prior-art WAGO-I/O-PRO (1999) and CoDeSys 2.2 (2001) products disclose this functionality as described on pages 1-4, 3-3, 3-54, 6-5 and 6-15 of the WAGO-I/O-PRO (1999) user's guide and on pages 2-27, 4-1, 4-4, 4-71 and 5-28 of the CoDeSys 2.2 (2001) user's guide.

166. To the extent an access tool may include a browser that interacts with the one or more industrial control components, the prior-art CoDeSys 2.2 (2001) product discloses this functionality as described on page 6-66.

167. Furthermore, I have found no evidence the WAGO-I/O system includes a database application (and Zatarain does not point specifically to one).

Analysis Claim 5

168. **The system of claim 2, the network includes at least one of a local factory network, a wireless network, and public network.**

169. As discussed, the WAGO-I/O system does not meet the limitations of Claim 2 and thus does not meet the limitations of this dependent claim. To the extent a network may include at least one of a local factory network, a wireless network, and public network, the prior-art WAGO-I/O-PRO (1999) and CoDeSys 2.2 (2001) products disclose this functionality as described on pages 6-5 of the WAGO-I/O-PRO (1999) user's guide and on page 4-71 of the CoDeSys 2.2 (2001) user's guide.

Analysis Claim 6

170. **The system of claim 2, the recording component logs interaction data that has been directed to the one or more industrial control components during a current application session associated with the access tool.**

171. As discussed, the WAGO-I/O system does not meet the limitations of Claim 2 and thus does not meet the limitations of this dependent claim.

172. Zatarain points to the CoDeSys 2.3 (2007) logging functionality to support the contention that CoDeSys meets the requirements of Claim 6. As discussed, the prior-art CoDeSys 2.2 (2001) and CoDeSys 2.3 (2007) disclose the same functionality.

Analysis Claim 9

173. **The system of claim 1, the tracking component aggregates activities logged by the recording component in at least one of a local storage and a remote storage location.**

174. As discussed, the WAGO-I/O system does not meet the limitations of Claim 1 and thus does not meet the limitations of this dependent claim.

175. Furthermore, there is no tracking component and no notion in the WAGO-I/O system of aggregation functionality. It simply includes logging functionality as was common for industrial control systems of the time.

Analysis Claim 10

176. **The system of claim 9, the tracking component aggregates transaction data by creating at least one of a file, schema, and a data structure in the local or remote storage locations, and tags the file, schema, and data structure with an identifier relating to the one or more industrial control components that have been accessed.**

177. As discussed, the WAGO-I/O system does not meet the limitations of Claim 9 and thus does not meet the limitations of this dependent claim.

178. Furthermore, there is no tracking component and no notion in the WAGO-I/O system of aggregation functionality. It simply includes logging functionality as was common for industrial control systems of the time.

Analysis Claim 14

179. **The system of claim 1, at least one of the recording component and the tracking component are employed to generate an audit report that describes interactions that have occurred with the one or more industrial control components.**

180. As discussed, the WAGO-I/O system does not meet the limitations of Claim 1 and thus does not meet the limitations of this dependent claim.

181. Furthermore, there is no tracking component and no notion in the WAGO-I/O system of an audit report. It simply includes logging functionality as was common for industrial control systems of the time.

Analysis Claim 16

182. **The system of claim 14, the audit report includes 1 to N fields, N being an integer, the fields displaying various types of auditing information.**

183. As discussed, the WAGO-I/O system does not meet the limitations of Claim 14 and thus does not meet the limitations of this dependent claim.

184. Furthermore, there is no notion in the WAGO-I/O system of an audit report or auditing information. It simply includes logging functionality as was common for industrial control systems of the time.

Analysis Claim 24

185. **Claim 24 - A method for verifying an industrial control process, comprising:**

186. As discussed, the WAGO-I/O system simply includes basic logging functionality. There are no functions directed towards verifying an industrial control process.

187. **monitoring activity data directed to one or more control components;**

188. Zatarain cites to the CoDeSys 2.3 (2007) manual to support his contention that CoDeSys meets this claim element. As detailed below, the prior-art CoDeSys 2.2 (2001) and WAGO-I/O-PRO (1999) products have the same monitoring functionality as CoDeSys 2.3 (2007).

Comparison of Monitoring Functionality in CoDeSys 2.2 (2001), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007)

189. The table below shows the introduction to the monitoring functionality in the CoDeSys 2.2 (2001), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007) manuals. Notice the text is exactly the same. CoDeSys 2.3 (2007) has the same monitoring functionality as CoDeSys 2.2 (2001) and WAGO-I/O-PRO (1999).

CoDeSys 2.2 (2001) user's guide (pages 2-27 – 2-28) *Monitoring* In Online mode, all displayable variables are read from the controller and displayed in real time. You will find this display in the declarations and program editor; you can also read out current values of variables in the watch and receipt manager and in a visualization. If variables from instances of function blocks are to be monitored, the corresponding instance must first be opened (see in this connection chapter 4.4: Create Objects).
WAGO-I/O-PRO (1999) user's guide (page 6-15) **6.3.4 Monitoring** In addition to the variables declaration visible on the screen the current values are constantly read out of the logic controller and displayed. The monitoring is always activated in simulation mode. The command '**Project**' '**Build**' **option** must be performed and '**Monitoring**' selected in the dialog appearing in order to be able to monitor with a Bus Controller.
CoDeSys 2.3 (2007) user's guide (page 2-24) **Monitoring** In Online mode, all displayable variables are read from the controller and displayed in real time. You will find this display in the declarations and program editor; you can also read out current values of variables in the watch and receipt manager and in a visualization. If variables from instances of function blocks are to be monitored, the corresponding instance must first be opened.
The same monitoring functionality was present in CoDeSys 2.2 (2001), WAGO-I/O-PRO (1999) and CoDeSys 2.3 (2007).

190. **tagging at least one file that is related to the one or more control components;**

191. As discussed, the WAGO-I/O system simply includes basic logging functionality. It includes no notion of tagging a file.

192. **logging the activity data in at least one of a local and remote location;**

193. Zatarain points to the CoDeSys 2.3 (2007) logging functionality to support the contention that CoDeSys meets this claim element. As discussed, CoDeSys 2.2 (2001) and CoDeSys 2.3 (2007) disclose this functionality.

194. **aggregating the logged activity data in the at least one file.**

195. As discussed, the WAGO-I/O system simply includes basic logging functionality. It includes no functionality for aggregating.

Analysis Claim 28

196. **Claim 28 - A audit system for an industrial control process, comprising: means for logging current access attempts for an industrial control process; means for at least one of communicating and categorizing the access attempts; means for storing the access attempts; and means for generating a report that details the access attempts that have occurred over time.**

197. Zatarain (page 74) states, "Other than the means plus function elements, I do not believe any of the claim terms in the '974 patent require explanation or a Court's interpretation, as the terms are clear and easily understood not only by a PHOSITA, but simply by laypeople as well," but provides no further discussion of the means plus function elements of the '974 patent. I note that the '974 patent does not describe the "means for . . . categorizing the access attempts" required by the claim, which leaves one of ordinary skill without essential information as to what means is used to categorize access attempts. In the event Plaintiffs assert contentions to this claim, I reserve the opportunity to address these contentions in a supplemental report.

Analysis Claim 29

198. Zatarain supports infringement contentions for this claim with the statement, "See infringement analysis above for the prior claims in the '974 patent." (Page 109) I was unable to locate the analysis Zatarain refers to and thus find the infringement contentions for this claim unsupported. As discussed below, I also find these infringement contentions incorrect and that CoDeSys 2.3 (2007) does not meet the limitations of this claim.

199. CoDeSys 2.3 (2007) saves log data to a binary file called "*.log" where the asterisk represents a file name that can be chosen by the user. Binary files do not have distinct "fields."

200. **Claim 29 - A computer readable medium having stored thereon a data structure, comprising:**

201. CoDeSys 2.3 (2007) saves log data to a binary file.

202. **a first data field representing real time access data to an industrial control component;**

203. As discussed, the binary format of CoDeSys 2.3 (2007) log file does not have "fields" and as such CoDeSys 2.3 (2007) does not meet this claim limitation.

204. **a second data field representing a tag name to store and aggregate the real time access data;**

205. As discussed, the binary format of CoDeSys 2.3 (2007) log file does not have "fields" and as such CoDeSys 2.3 (2007) does not meet this claim limitation. Furthermore, I find no tag names in the log file associated with aggregating data and Zatarain does not point to any.

206. **a third data field to categorize the real time access data.**

207. As discussed, the binary format of CoDeSys 2.3 (2007) log file does not have "fields."

[The remainder of this page is intentionally left blank. The signature page follows.]

IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF WISCONSIN

I declare under penalty of perjury that the foregoing Supplemental Declaration of Richard Hooper, Ph.D., P.E.is true and correct to the best of my knowledge and belief. Executed on March 22, 2012.

R. Hooper

Richard Hooper, Ph.D., P.E. (Signature)

APPENDIX A

List of References and Products

Appendix A

Materials

1. Manual for PLC-Programming with CoDeSys 1.5 (1997)
2. R.W. Lewis, Programming Industrial Control Systems Using IEC 11131-3, 169-187 (1998) (*attached*)
3. User Manual for PLC Programming with CoDeSys 2.2 (2001)
4. WAGO 758-870—WAGO-I/O-IPC-G2 Manual (2009)
5. WAGO 759-120—IEC 1131-3 Programming Tool, WAGO-I/O-PRO User Manual (1999)
6. WAGO 759-333—User Manual for PLC Programming with CoDeSys 2.3 (2007)

WAGO Products

7. 767-4801
8. 767-2301
9. 750-880
10. 750-872
11. I/O-IPC-G2 0758-0870
12. I/O-IPC 758-870
13. 750-842
14. 750-841
15. 750-600
16. 750-501
17. 750-410
18. 759-333
19. 759-302

IET CONTROL ENGINEERING SERIES 50

Series Editors: Professor D.P. Atherton
Professor G.W. Irwin

Programming Industrial Control Systems Using IEC 1131-3

Revised Edition

Published by The Institution of Engineering and Technology, London, United Kingdom

First published 1998
Reprinted 2007

The Institution of Engineering and Technology
Michael Faraday House
Six Hills Way, Stevenage
Herts, SG1 2AY, United Kingdom

www.theiet.org

British Library Cataloguing in Publication Data
A catalogue record for this product is available from the British Library

ISBN (10 digit) 0 85296 950 3
ISBN (13 digit) 978-0-85296-950-2

Printed in the UK by Short Run Press Ltd, Exeter

ndon, United Kingdom

nd Technology

Contents

Chapter 7

Instruction List

In this chapter we will review Instruction List; this is the fourth language of the IEC languages in the set:

Structured Text, Function Block Diagram, Ladder Diagram, Instruction List and Sequential Function Chart.

Instruction List is a low level language that can be used to express the behaviour of functions, function blocks and programs, and also actions and transitions in Sequential Function Charts.

In this chapter we will review:

- The basic structure of the Instruction List language;
- The behaviour of standard operators;
- How to control the flow of program execution using conditional operators, jumps and labels;
- How to call function blocks and functions in IL;
- Guidelines on writing IL;
- Restrictions on the portability of code between IL and the other IEC languages.

7.1 IL background

Instruction List is a low level language which has a structure similar to a simple machine assembler. The IEC has developed Instruction List by reviewing the many low level languages offered by a wide range of PLC manufacturers. The IL language, as defined in IEC 1131-3, provides a range of operators that represent those most commonly found in proprietary instruction list languages of current day PLCs.

The basic structure of Instruction List is very simple and easy to learn. It is ideal for solving small straightforward problems where there are few decision points and where there are a limited number of changes in program execution flow.

A number of the manufacturers offering IEC 1131-3 based PLCs have chosen to support IL in preference to Structured Text. For a PLC designer, one of the main advantages of IL over ST is that IL is far easier to implement. It is fairly straightforward to develop a PLC that can interpret IL instructions directly. With some systems it is possible to download IL programs to a PLC without going through a program compile and build process. In contrast, Structured Text generally has to be compiled to the native micro-processor assembler of the PLC, such as an 80486 assembler, before it can be down-loaded and run.

Instruction List is sometimes regarded as the language into which all the other IEC languages can be translated. It is the base language of an IEC compliant PLC, into which all other languages such as Structured Text and Function Block Diagram can be converted. However, there are implementors who have taken a different view, and tend to regard IL as a language strictly for small PLCs and treat Structured Text as the base language.

It should be emphasised that the IEC 1131-3 standard does not imply that any particular language should be treated as the base language. Instruction List is merely another language which can be used, if preferred, for solving certain control problems.

As is found when using any machine assembler language, there are a number of disadvantages to using Instruction List over using a high level language such as Structured Text. For example, it is not possible to check the structure of a section of an IL program with the same rigour as a section written in ST. It is also harder to follow the program flow in IL than in ST.

Perhaps one advantage of IL is that, in some circumstances, it may be used to write tight, optimised code for performance critical sections of a program.

7.2 IL language structure

Instruction List is a language which consists of a series of instructions where each instruction is on a new line. An instruction consists of an operator followed by one or more operands. An operand is the 'subject' of the operator. Operands represent variables or constants as defined in Chapter 3, 'Common elements'. A few operators can take a series of operands, in which case each operand should be separated by a comma.

Each instruction may either use or change the value stored in a single register. The standard refers to this register as the 'result' of an instruction. The 'result' can be overwritten with a new value, modified or stored in a variable. The result register is sometimes alternatively referred to as the accumulator or 'accu'.

IL also provides comparison operators which can compare the value of a variable with the register value, i.e. with the current result. The result of the comparison is written back to the register. Various types of jump instruction are provided that can test the current register value and, if appropriate, jump to a named label. Labels can be used to identify various entry points for jump instructions. Each label should be followed by a colon.

The following figure depicts the main features of the IL language:

```
Label        Operator   Operand      Comment

             LD         Speed        (* Load Speed and *)
             GT         1000         (* Test if > 1000 *)
             JMPCN      VOLTS_OK     (* Jump if not *)
             LD         Volts        (* Load Volts and *)
             SUB        10           (* Reduce by 10 *)
             ST         Volts        (* Store Volts *)
VOLTS_OK:    LD         1            (* Load 1 and store *)
             ST         %Q75         (* in output 75 *)
```

Figure 7.1 IL language structure

The IL instructions in Figure 7.1 are equivalent to the following Structured Text statements:

```
IF Speed > 1000 Then
 Volts := Volts - 10;
END_IF;
%Q75 := 1;
```

Consider the first line of the IL example; LD represents the load operator, and Speed is the operand and identifies the value of an integer variable. The LD instruction loads the current value of variable Speed into the result register.

The 'greater than' GT operator compares the value of the register with literal (constant) 1000. If the value of the register is greater than 1000, the result of the GT operator will be 1 which is equivalent to a boolean true value. Otherwise it will be set to 0, a boolean false value.

It is important to note that the value of Speed is no longer available in the register and that the datatype of the value in the register has changed from integer (INT) to boolean (BOOL).

The JMPNC instruction jumps to the label VOLTS_OK if the result register contains 0, i.e. when the comparison fails.

If the comparison is successful, the jump is not made and the instructions that follow are executed. In this case, the value of variable Volts is loaded into the register. The SUB operator subtracts 10 from the register. The ST operator then stores the result back to variable Volts.

The last two instructions load 1 into the 'result' register and then store the register value into the output 75.

In order to improve readability of the language, IL instructions are structured so that labels, operators and operands appear in fixed tabulation (tab) positions.

7.3 Comments

Instruction List comments have the same format as in Structured Text. However, with IL, a comment can only be placed at the end of the line. It is not acceptable for a comment to appear at the beginning of a line or between an operator and an operand.

One or more blank lines can be inserted between instructions to break up long sequences of instructions and aid readability.

7.4 Instruction semantics

The standard states that the general behaviour or semantics of an IL instruction for the majority of operators can be described by the following expression:

```
NewResult := CurrentResult Operator Operand
```

'CurrentResult' represents the value currently held in the 'result' register prior to the instruction being executed; it is the value left over from the previous instruction.

'NewResult' is the new value to be loaded into the 'result' register as produced by executing the instruction. It is the value produced by the 'Operator' acting on values of 'CurrentResult' and the 'Operand'. The 'NewResult' becomes the 'CurrentResult' for the next instruction.

ger available in the
hanged from integer

if the result register

the instructions that
s is loaded into the
he ST operator then

and then store the

ons are structured so
tab) positions.

ured Text. However,
. It is not acceptable
an operator and an

ns to break up long

an IL instruction for
pression:

or Operand

ult' register prior to
from the previous

register as produced
'Operator' acting on
esult' becomes the

For example, consider the instruction 'SUB 10' in Figure 7.1 that uses the Subtract operator. This is equivalent to the expression:

```
NewResult := CurrentResult SUB 10
```

The SUB operator takes 10 from the 'CurrentResult' and stores the value in the 'NewResult'.

With comparison operators, the 'CurrentResult' is compared with the operand. The 'NewResult' is set to 1 (true) if the comparison is successful; otherwise it is set to 0 (false).

For example, the behaviour of the comparison instruction 'GT 1000' in Figure 7.1 that uses the 'greater than' operator, is equivalent to:

```
NewResult := CurrentResult GT 1000
```

A few IL operators do not fit this general model. For example, the ST (store) operator can be described by the expression:

```
Operand := CurrentResult
```

This is because the function of the ST operator is to store the current value held in the result register into the location defined by the Operand field.

Tables 7.1 and 7.2 describe the standard IL operators. Additional text and notes are given with the tables to describe the operator behaviour.

Deferred execution

Special modifiers called the parenthesis (or bracket) modifiers allow sections of IL instructions to be deferred, i.e. to produce intermediate results that do not affect the current 'result register'. This has the same effect as the use of brackets in normal arithmetic and boolean expressions.

For example:

```
LD   A    (* Add A to B*)
ADD  B    (* hold the value in result reg. *)
MUL( A    (* Defer MUL until (A-B) available *)
SUB  B
)         (* Now multiply by (A-B) *)
```

In this example, when the MUL operator is reached, the result of A + B is held in the result register. The left bracket '(' following the MUL operator indicates that the multiply operation will be deferred until the right bracket ')' operator is reached. The value of operand A is then loaded into a temporary result register.

The SUB B instruction produces the value of A - B, which is held in the temporary result register. The final ')' operator takes the value in the temporary

result register and multiplies it by the value held in the main result register. This sequence of instructions is equivalent to:

```
result := (A + B) * (A - B);
```

Note: There is no precedence with IL operators. Hence the brackets are needed in this expression, i.e. around (A + B), to indicate that the add operator occurs before the multiplication operator.

The parenthesis operators provide a similar function to a stack. A number of deferred operators can be active at any time allowing fairly complex nested operations to be performed.

Consider the following example:

```
LD     X     (* Load X                  1 *)
ADD(   B     (* Defer ADD, load B       2 *)
MUL(   C     (* Defer MUL, load C       3 *)
ADD    D     (* Add D                   4 *)
)            (* Multiply result         5 *)
)            (* Add result              6 *)
```

This is equivalent to:

```
X + (B * ( C + D))
```

The following table shows how the deferred instructions in the example cause values to be stored in a stack of result registers. The table gives the values in the top three result registers that exist after each instruction in this example has been evaluated.

Instruction	Top result register	Top -1 result register	Top-2 result register
1	X		
2	X	B	
3	X	B	C
4	X	B	C+D
5	X	B*(C+D)	
6	X+B*(C+D)		

Note : The standard does not indicate that the use of jump instructions within parenthesised sections is illegal. Clearly, in practice, jumps from within parenthesised instructions may produce unpredictable results and should be avoided.

Modifiers

Some IL operators can take single letter modifiers after the operator mnemonic that change the semantics of the instruction.

If the operand contains a boolean value, the 'N' modifier can be used to negate its value. With jump operators, the 'N' modifier indicates that the jump will use the negated value of the 'result register'.

The 'C' modifier can only be used with jump operators and indicates that a jump is made conditionally on the boolean value of the result register.

For example:

```
LD      %IX10     (* Load Input 10 *)
ANDN    Switch1   (* AND NOT switch1 *)
JMPNC   Lab1      (* Jump if Not true *)
```

The ANDN operator inverts the value of Switch1 and ANDs the result with the value of input 10 (%IX10). The JMPNC jump operator jumps to label 'Lab1' if the result is not boolean true.

7.5 Operators

The main IL operators are listed in Table 7.1; the comparison operators and instructions concerned with changing the flow of execution control are listed in Table 7.2.

Table 7.1 Main IL operators

Operator	Modifiers	Operand	Comments
LD	N	ANY$_1$	Load operand into result register
ST	N	ANY$_1$	Store result register into operand
S	Note 2	BOOL	Set operand true
R	Note 2	BOOL	Reset operand false
AND	N,(	ANY$_1$	Boolean AND
&	N,(	ANY$_1$	Boolean AND (equivalent to AND)
OR	N,(	ANY$_1$	Boolean OR
XOR	N,(	ANY$_1$	Boolean exclusive OR
NOT		ANY$_1$	Logical negation (one's complement)
ADD	(	ANY$_1$	Addition
SUB	(	ANY$_1$	Subtraction
MUL	(	ANY$_1$	Multiplication
DIV	(	ANY$_1$	Division
MOD	(	ANY$_1$	Modulo-division

Table 7.2 IL comparison and jump operators

Operator	Modifiers	Operand	Comments
GT	(	ANY [1]	Comparison greater than
GE	(	ANY [1]	Comparison greater than, equal
EQ	(	ANY [1]	Comparison equal
NE	(	ANY [1]	Comparison not equal
LE	(	ANY [1]	Comparison less than, equal
LT	(	ANY [1]	Comparison less than
JMP	C,N	LABEL	Jump to label
CAL	C,N	NAME	Call function block, see following sections on the alternative call syntax
RET	C,N		Return from function or function block (see Note 4)
)			Execute the last deferred operator

Note 1: Operators that can take operands of data type ANY are said to be 'overloaded'. That is, the operator can be used with any of the different elementary data types such as integers SINT, INT, time data types such as DATE_AND_TIME, floating point values of type REAL and so on. For a full list of elementary data types see Chapter 3, 'Common Elements'. The operator will only produce a valid result if the current value in the result register is of the same data type as the operand.

Note 2: The Set and Reset operators can only be used with operands of boolean data type.

Note 3: Operators that can have more than one modifier may be used with either or both modifiers. For example there are four forms of the AND operator:

Operator	Semantics
AND	Boolean AND
AND (	Deferred boolean AND
ANDN (	Deferred boolean AND, invert deferred result
ANDN	Invert boolean AND

Note 4: The RET operator is used to return from a function or function block. The modifier C means that the return is made conditionally. If the result register contains boolean true, i.e. 1, the return is made. The N indicates that the result register should be inverted, i.e. false for the return to be made.
With JMP, CAL and RET instructions, the negate 'N' modifier can only be used with the condition modifier 'C', and indicates that the jump is made conditionally on the negated value of the result register, i.e. when the value is boolean 0 (false).

7.6 Calling functions and function blocks

Within Instruction List, the standard provides three different formats of the CAL operator for calling function blocks, and two alternative formats for calling functions. The syntax for calling functions and function blocks is not the same.

Calling a function block using a formal call with an input list

With this format, the CAL operator is followed by a list of function block parameters with their values. Each input parameter must be identified by name. The value of each parameter can be given directly or calculated and passed into the IL current result register (accumulator).

Example:

```
CAL    LOOP1(
SP:= 300.0
PV :=(
LD %IW20
ADD 10
)
)
```

This calls function block instance LOOP1 with parameter SP set to 300.0 and parameter PV set to the value of input word 20 plus 10.

Calling a function block using an informal call

In this format, the values to each input should be set-up prior to calling the function block.

Example:

```
LD    300.0
ST    LOOP1.SP
LD    %IW20
ADD   10
ST    LOOP1.PV
CAL   LOOP1
```

This is equivalent to the previous example; LOOP1 is called with parameter SP set to 300.0 and parameter PV set to the value of input word 20 plus 10.

Calling function blocks using input operators

This format can only be used with some of the IEC standard function blocks as defined in Chapter 9. A number of IL operators are reserved for use with commonly used function blocks such as the SR (Set/Reset) bistable, or the CTU up-counter. A full list of such operators is given in the next section. They are provided as a short-hand form for calling commonly used function blocks.

Example:

```
S1    Latch1
LD    10
PV    CTU1
CU    CTU1
```

The S1 operator can only be used with an SR bistable. In this case the S1 instruction causes Latch1, a function block instance of type SR, to be set.

The PV (preset value) operator can only be used with counter function blocks. In the example, the PV instruction loads 10 from the result register into the PV parameter of the up-counter function block CTU1. The CU instruction calls the up-counter function block with the counter enable parameter set true, i.e. to start counting.

Note: Parameters that are not supplied in a function block call will either take the previous value they were assigned, or take their default value from initialisation.

Calling functions using a formal call

In a formal function call, the function name is given, followed by the values of the input parameters.

Example:

```
SHR(
IN := %IW30
N:= 10
)
```

An amendment to the 1993 edition of IEC 1131-3 for the second edition proposes that when a formal call is made to a function, then optionally the value of the first parameter can be supplied directly from the current result.

Example:

```
LD    %IW30
SHR(
N:= 10
)
ST    %QW100
```

Calling functions using an informal call

In an informal function call, the function name is given, followed by the values of the input parameters. The value of the first parameter is supplied by the current result.

Example:

```
LD    %IW30
SHR   10
ST    %QW100
```

The SHR (shift right) function requires two input parameters. In this example, the first parameter is loaded from the current result, i.e. the contents of input word 30; the second parameter has value 10, i.e. the count of bits for the shift operation. The result from the function is returned as the current result. This is loaded to output word 100.

7.7 Defining functions and function blocks

Instruction List can be used to define the behaviour of functions and function blocks. When used to define a function, the value returned by the function is provided by the last value in the result register (accumulator).

When used to define function blocks, IL can reference the function block inputs (VAR_INPUT) and write values to outputs (VAR_OUTPUTS) that are declared in the function block declaration.

For further details on using IL to define functions see Section 7.12 IL program example'.

7.8 Function block operators

The full list of reserved operators for use with standard function blocks is given in Table 7.3.

Table 7.3 IL Function block operators

Operator	Function block type	Comments
S1,R	SR bistable	Set and Reset the SR bistable.
S,R1	RS bistable	Set and Reset the RS bistable.
CLK	R_Trig, rising edge detector	Clock input to the rising edge detector function block.
CLK	F_Trig, falling edge detector	Clock input to the falling edge detector function block.
CU,R,PV	CTU, up-counter	Control parameters for the CTU up-counter function block, to count-up (CU), reset (R) and load (PV).
CD,PV	CTD, down-counter	Control parameters for the CTD down-counter, to count-down (CD), load LD and set the count minimum (PV).
CU,CD,R, PV	CTUD, up/down-counter	Control parameters; same as for up- and down-counters.
IN,PT	TP, pulse timer	Control parameters for the pulse timer, to start timing (IN) and set-up the pulse time (PT).
IN,PT	TON, on delay timer	Control parameters for the on-delay timer, to start timing (IN) and set-up the delay time (PT).
IN,PT	TOF, off delay timer	Control parameters for the on-delay timer, to start timing (IN) and set-up the off-delay time (PT).

7.9 Deficiencies and proposed extensions

Of all the five IEC languages, the definition of the Instruction List language has been found to be the most contentious. The semantics, i.e. the way the instructions operate, are not always fully described in the standard.

One of the main problems is that the standard does not fully define the behaviour of a conceptual processor, or virtual machine, that can execute IL instructions. For example, it is unclear how the result register stores values of different data types. It is particularly difficult to see how the accumulator can be

able.
able.
edge detector
edge detector
e CTU up-
count-up
PV).
e CTD down-
CD), load LD
m (PV).
as for up- and
e pulse timer,
t-up the pulse
e on-delay
and set-up the
e on-delay
and set-up the
ist language has
the instructions
fully define the
can execute IL
stores values of
umulator can be

used to store multi-element variables, such as structured variables, arrays or strings.

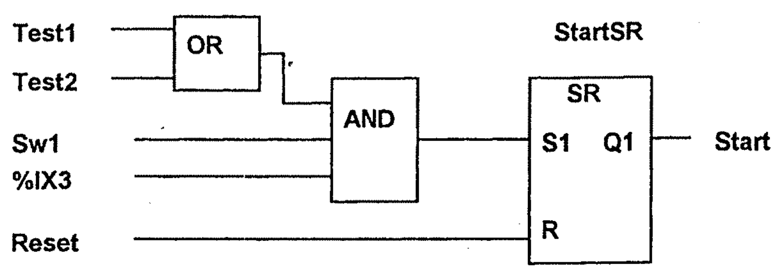
The run-time behaviour for various error conditions is also not documented. For example, the behaviour following the use of an operator with an inappropriate data type is not described.

7.10 Portability between IL and other languages

The conversion of sections of Instruction List into the other languages is very difficult. It could be achieved if a restricted range of IL operators were used and the instructions written using a strict format.

Conversion of the other languages into IL is easier but by no means always straightforward.

For example consider the following section of a Function Block Diagram:



This can be written in Instruction List as follows:

```
LD    Test1        (* Test1 OR *)
OR    Test2        (* Test2 *)
AND   Sw1          (* AND Sw1 *)
AND   %IX3         (* AND input 3 *)
ST    StartSR.S1   (* Set input of StartSR *)
LD    Reset        (* Load value of Reset *)
ST    StartSR.R    (* Store in reset input *)
CAL   StartSR      (* Call fb. StartSR *)
LD    StartSR.Q1   (* Load output Q1 *)
ST    Start        (* and store in Start *)
```

The execution control parameters EN and ENO used in the FBD and LD languages can be used in IL. They are treated as extra parameters of a function or function block.

7.11 Good programming style

Because Instruction List is a fairly low level language, great care should be taken in laying out the code to ensure that it is easy to read and maintain. It is important that IL instructions are well commented. Jump instructions should be used sparingly so that the flow of execution through the code is easy to follow.

Particular care should be taken to ensure that the value in the result register is always appropriate for an IL operator.

7.12 IL program example

In this example we will consider the use of IL for defining a function for calculating the travel distance between two points on a flat surface. Such a function might be required for controlling a numerically controlled (NC) machine that needs to traverse between two drilling points. Assume that the positions of the two points are given by x and y co-ordinates as shown in Figure 7.2.

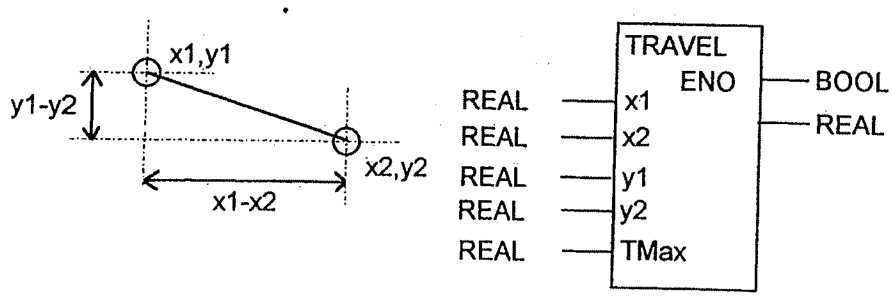


Figure 7.2 IL function example

From trigonometry, the distance between the two points can be expressed in Structured Text as:

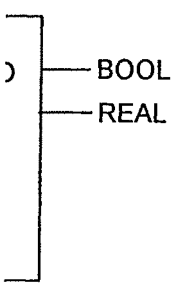
```
Travel_distance := SQRT (( X1-X2)*(X1-X2)
                        + (Y1-Y2)*(Y1-Y2));
```

Assume that this function will be used in the graphical languages. The ENO output signal is therefore required for passing execution control on to other functions when the function has completed successfully, see Chapter 5, Section 5.

If the travel distance is less than TMax, the function should set ENO to indicate that it has computed the travel distance successfully. If the travel distance exceeds

hould be taken
It is important
iould be used
ollow.
esult register is

a function for
urface. Such a
(NC) machine
positions of the
..



be expressed in

:)
2));

iges. The ENO
ol on to other
er 5, Section 5.
ENO to indicate
istance exceeds

TMax, the computation has produced a travel distance outside the range of the controlled machine, in which case the ENO output should not be set.

The full definition of a function TRAVEL( ) to calculate the required movement expressed using Instruction List is as follows:

```
FUNCTION TRAVEL : REAL
 VAR_INPUT
   X1,X2,Y1,Y2 : REAL; (* X and Y Co-ordinates *)
   TMax    : REAL:     (* Maximum travel distance *)
 END_VAR
 VAR
   temp : REAL;   (* Intermediate values *)
 END_VAR

     LD   Y1
     SUB  Y2     (* Subtract Y2 from Y1 *)
     ST   Temp   (* Store Y1-Y2 in Temp *)
     MUL  Temp   (* Multiply by Temp to square *)
     ADD( X1     (* Defer ADD *)
     SUB  X2     (* Subtract X1 from X2 *)
     ST   Temp   (* Store X1-X2 in Temp *)
     MUL  Temp   (* Multiply by Temp to square *)
     )           (* Add two squared values *)
     CAL  SQRT   (* Call Square root fun. *)
     ST   TRAVEL (* Setup function result*)
     GT   TMax   (* Greater than TMax ? *)
     JMPC ERR    (* Yes, Jump to Error *)
     S    ENO    (* Set ENO *)
     RET         (* Normal return *)

 ERR:
     RET        (* Error return, ENO not set *)

END_FUNCTION
```

Note: The value for the SQRT function is supplied from the current result, as discussed in Section 7.6.

Summary

Instruction List is a low level textual language which can be used for describing the behaviour of functions and function blocks, and actions and transitions used in the Sequential Function Charts.

- It is simple to implement and can be directly intepreted within a PLC. It has therefore been adopted by a number of PLC manufacturers for their small to medium sized PLCs.
- It can be used to write tight, optimised code for performance critical operations.
- Operators are provided to manipulate variables of all the elementary data types, call functions and function blocks.
- It is ideal for solving small straightforward problems where there are few decision points and where there are a limited number of changes in program execution flow.
- Care should always be taken when developing programs using IL as there are only limited language validation checks that can be undertaken on a programming station. The majority of language violations, such as data type inconsistencies, can only be detected at run-time.
- Conversion from IL into the other IEC languages is not always possible. However, conversion from the other languages into IL can generally be achieved.